author of the configuration file to ensure that the assembly targeted by the redirect can be found. In the case of the application configuration file, the application author/deployer may choose to package the files (e.g., binaries) for the target assembly with the application, or alternatively provide information (sometimes referred to as a "codebase") in the configuration file to tell the operating system where the files can be found. Another option is to install the redirected version of the assembly on the machine. Note that for privatized assemblies, e.g., in the second alternative mode, by attempting to locate the referenced assembly by consistently looking in the application directory, an application author may simply copy newer privatized assemblies into the application directory, and know that the newer assemblies will be automatically used by the application.

[0074] As generally described above, a second alternative mode is provided, in which the order of applying configurations is different, a safe mode is available for bypassing publisher configuration, and an administrator configuration may be present and if so is interpreted to override other configuration binding data. More particularly, in this second mode, the first stage of policy resolution comprises resolving any application policy. As represented in **FIG. 3B** by circled numerals four (4) and five (5), if an application policy exists, a binding mechanism **305** reads and interprets the application policy **216**. To this end, before a bind to the assembly can proceed, the application policy file (if any) is accessed, and analyzed. As described above, if not bypassed via the safe mode, any publisher configuration is next applied, as generally represented in **FIG. 3B** by the arrows labeled six (6) and seven (7).

[0075] In this second mode, a third stage in the bind configuration resolution process is administrator configuration, represented in **FIG. 3B** by the arrows labeled eight (8) and nine (9). Administrator configuration is the strongest form of configuration, as it makes the final determination as to which version will be bound, and cannot be bypassed. To provide administrator configuration, the administrator configuration file **224 (FIG. 2B)**, e.g., named "machine.config," has the same schema as the configuration files used in the two previous stages of configuration resolution. Administrator configuration affects assembly binds that occur to any application on the system.

[0076] Once the configurations have been handled for a given assembly to which an application wants to bind, only one version of that assembly remains. This information may be cached in the activation context **302** (the arrow labeled eight (8) in **FIG. 3A** or ten (10) in **FIG. 3B**) and may be persisted (e.g., in the first alternative mode) so that it need not be computed again, unless and until a configuration change occurs. Note that in the second alternative mode, the application context is not persisted, although it is feasible to do so.

[0077] By way of example, **FIG. 4** represents some of the information that may be maintained in an activation context, e.g., an activation context **302** constructed for the application $\mathbf{200}_1$. In **FIG. 4**, the activation context **302** includes a table of contents **400** (e.g., providing offsets to its record-sets) for rapid access to the data therein.

[0078] For persisted activation contexts, because the configurations that may change the dependency information may change over time, (e.g., publisher configurations may be wrapped as assemblies which can be versioned, new application configurations may be downloaded and so on), in the first alternative mode the activation context **302** includes a cache coherency section **402**. The cache coherency section is used to detect whether a saved activation context **302** is valid, wherein when the activation context **302** is not coherent with current configuration, it is recomputed. A section per API that implements version-specific binding is maintained.

[0079] To map the application's requests to the proper assembly versions, the activation context **302** includes a DLL redirector section **404** and an object class redirector section **406**. The DLL redirector section **404** includes a record or the like for each DLL dependency that includes fields (e.g., **408**, and $\mathbf{409}_1$) that relate the DLL name used by an application to the exact pathname of the version determined following the above-described configuration-resolution process. The object class redirector section **406** includes a record or the like for each object class (e.g., Windows® object class) on which an application depends, wherein each record includes fields (e.g., $\mathbf{412}_1$, $\mathbf{413}_1$, and $\mathbf{414}_1$) that relate the object class name used by an application to the DLL file it is in and a version specific name. Note that the fields are arranged in a manner that optimizes lookup, e.g., the application-provided request data corresponds to the search key, and the records may be arranged in any way (e.g., alphabetically, by frequency, linearly, binary or so on) to speed searches.

[0080] Via the activation context, during runtime as described below, an application's requests for assemblies can be efficiently satisfied with the correct version of that assembly. If a given assembly is not found in the activation context data, the default assembly is used. To summarize, when an application first runs, the activation context built from the manifest data is cached, whereby the global, version-independent named objects requested by an application are mapped to version-dependent named objects as specified in the manifest and redirected by any configurations. As the application executes and requests a named object via one of the activation APIs, the version-independent named objects are applied in a version-specific fashion by accessing the application context, whereby the application gets the correct version.

[0081] **FIG. 5** represents the general operation during runtime, e.g., in the first alternative mode, wherein an application **200** requests via one of a set of application APIs (e.g., $\mathbf{300}_1$) a version-independent assembly (represented via the arrow labeled with circled numeral one (1)). Application APIS include those directed to loading DLLs, COM server loading, COM interface proxy stub metadata, type libraries for COM, program identifiers for COM, object (e.g., Windows®) classes, kernel global objects (e.g., semaphores, mutexes, events, shared memory, COM categories), application settings and database (registry) connections.

[0082] When the application API $\mathbf{300}_1$ receive the request, the request data (e.g., the application provided name) is passed to a runtime version-matching mechanism **500** (the arrow labeled two ($_2$)). The runtime version-matching mechanism **500** locates the correct activation context **302** (from among a store **502** or the like of those maintained) for the calling application **200**, and accesses the records therein to determine the correct version of the requested assembly